El renderizado estilo anime de Guilty Gear Xrd

Programa 12 de @CodigoPodcast

Por David Gallardo @galloscript y David Ramos @dramos_malaga

Referencias

- Galerías de imágenes
    - http://www.4gamer.net/games/216/G021678/20140714079/screenshot.html?num=0
    - http://www.4gamer.net/games/216/G021678/20140703095/screenshot.html?num=011
- Hilo de polycount respecto a este tema
    - http://www.polycount.com/forum/showthread.php?t=121144

¿Que es Guilty Gear Xrd?

- Videojuego de lucha 2D
- Desarrollado por la empresa Arc System Works con el motor Unreal Engine 3,
- Quinto de la saga Guilty Gear y fue
- Publicado como Arcade y más tarde en PS3 y PS4
- No se ha publicado en España, pero se puede comprar por internet.

La historia de porque este contenido

- Hace un tiempo, cuando salió el trailer de Guilty Gear Xrd un compañero nos los pasó en el trabajo y discutimos un poco si eso era 2D, 3D, como era el cel shading etc, y bueno mas tarde me documenté un poco sobre este tema.
- Que diferencia hay con otros juegos del mismo estilo? Probablemente el mejor cel shading hasta antes del Guilty imitando Anime es el que habíamos visto en la serie de Naruto Shippuden: Ultimate Ninja Storm, pero para mi en Guilty le dan una vuelta de tuerca más, imitando casi a la perfección sprites animados a mano.
- En el último Arcadia Gamers (266) hicieron un análisis de ese juego, me gustó bastante, y justamente mencionan esta reacción que tiene mucha gente al ver ese 2D/3D.
- Como tenía material de documentación guardado decidí hacer este contenido.

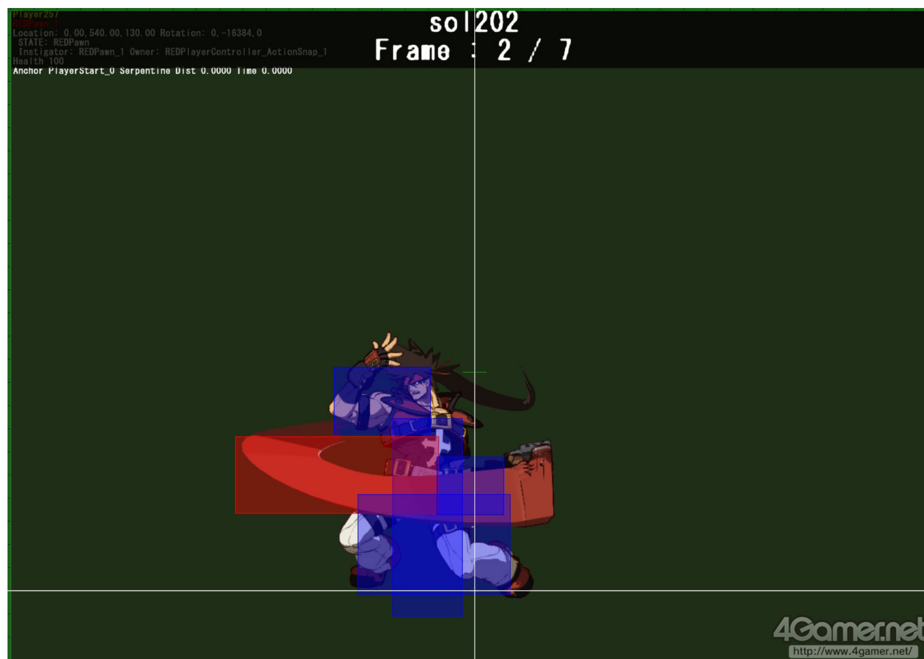
La documentación

- Es una suerte, por que casi todo el material de documentación oficial está en japones y alguien tuvo la bondad de traducir parte de este material en los foros de PolyCount.com
- También me doy cuenta de la cantidad de publicaciones respecto a estos temas que hay fuera de nuestro país y aquí son prácticamente inexistentes, es increíble encontrar una revista que te table de las bondades de unreal engine 4 o unity5 como developer, pero frustrante que solo esté en japonés.
- La mayoría de información está sacada de un artículo traducido de la página 4gamer, cuyo autor es Zenji Nishikawa. La traducción está disponible en los foros de polycount.com por el usuario Chev.

Desarrollo

- El juego fue desarrollado en Unreal Engine 3, aprovechando las rebajas en la licencia que supuso la salida de UE4, y sobre todo porque el equipo de programadores era bastante limitado, y necesitaba una herramienta completa y con buen soporte, además del scripting que era muy útil para que perfiles de no programadores modificaran el juego.
- El primer concept art del juego se realizó en 2008, y la producción empezó en 2011. Empezaron con un video prototipo para ver que tal resultaban los gráficos 3D en lugar del que habían utilizado hasta ahora, y a mitad de 2012 se pasó a full production del juego hasta finales de 2013.
- Hay que decir que llevaban experimentando con utilizar 3D desde el 2007, pero hasta ese prototipo no consideraban que fuera suficientemente expresivo, sobre todo en resoluciones en pantalla bajas.
- El equipo principal estaba compuesto por 25 personas, principalmente 4 programadores, 3 game designers encargados también de la planificación y 12 artistas y entre otros roles, además hay que sumar alrededor de un centenar de freelance en los que se externalizó parte del trabajo, supongo que artístico.

- Inicialmente el trabajo consiste en portar parte de lo que ya tenían implementado en anteriores motores para que sus herramientas que funcionaran con unreal engine 3.
    - Por ejemplo, para los juegos de lucha como este en 2D, es habitual tener una herramienta para sincronizar hit boxes con las animaciones.
    - Este tipo de herramientas tan concretas no estaban en ue3, pero esta gente las tenían desarrolladas y lo que hacían era dar soporte a su formato e implementar el runtime que lo utiliza en el juego.

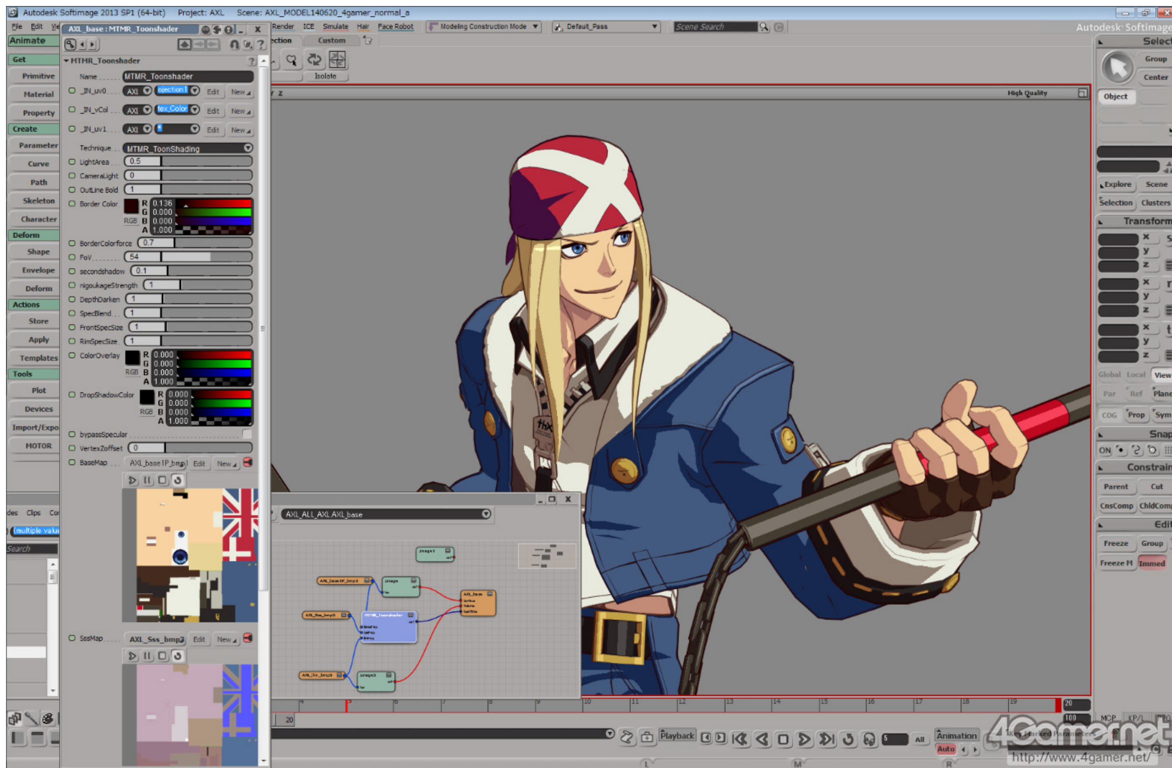


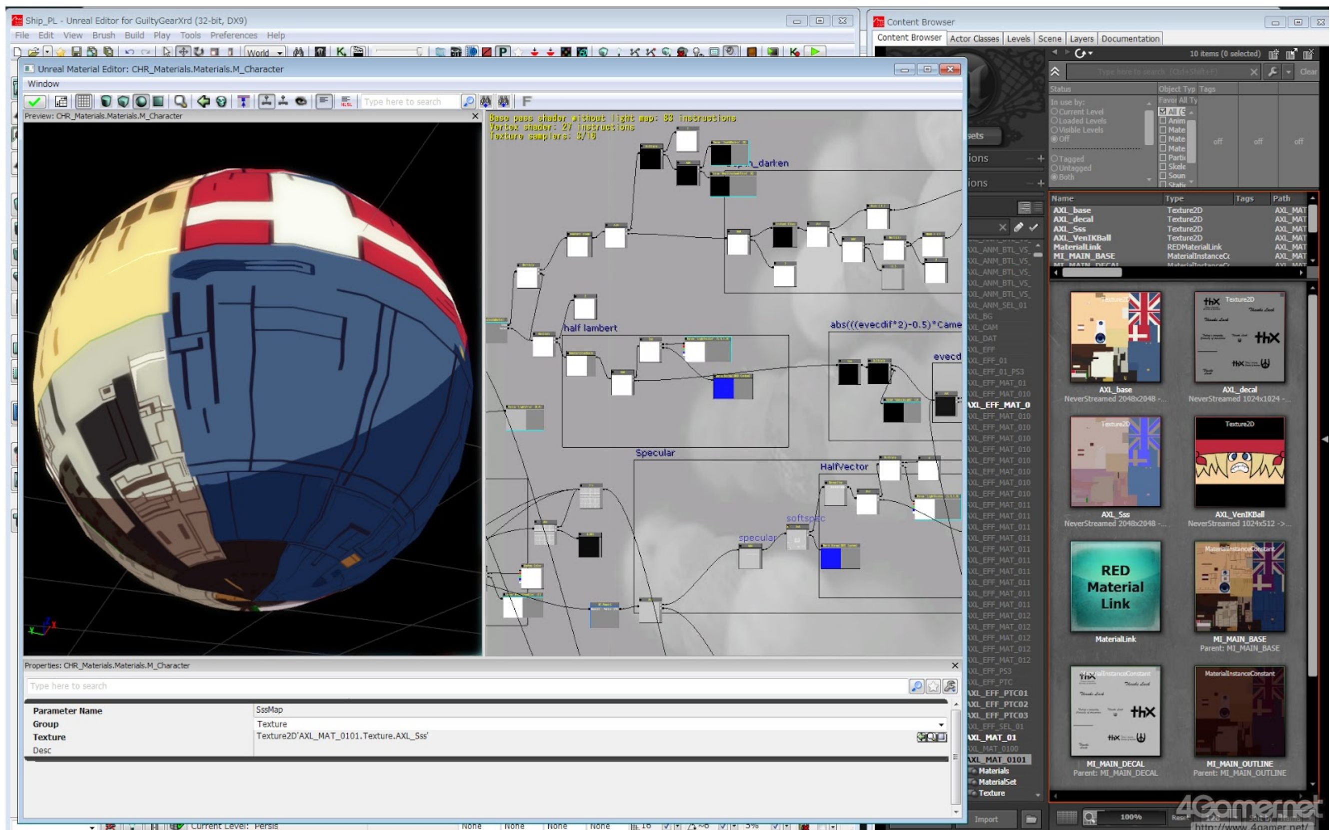
Técnicas

- Ahora empezamos con la parte buena, supuestamente todo lo que vamos a comentar a continuación está hecho al servicio de imitar el anime. Hay técnicas muy populares que todos conocemos, pero otras de las que no he había oído hablar nunca.

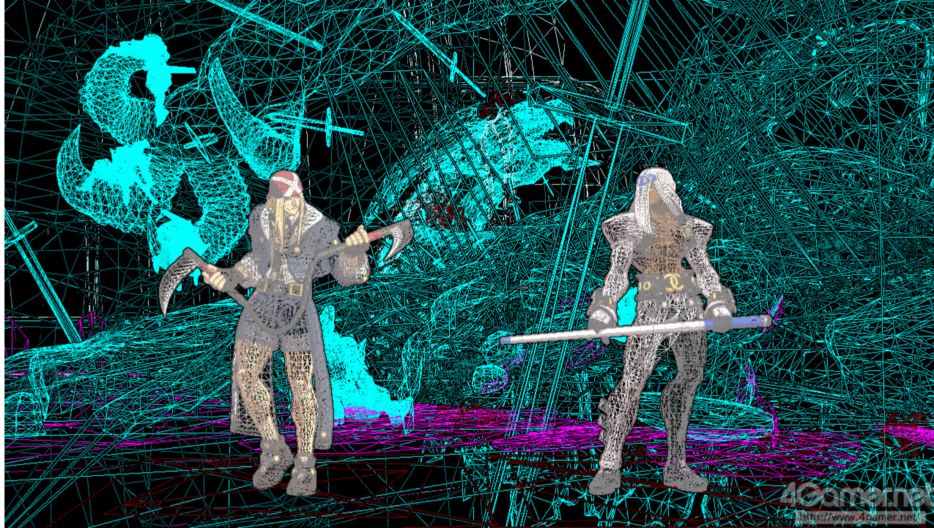
Modelado 3D

- Los modeladores 3D utilizaban Softimage y para conseguir un máximo WYSIWYG “what you see is what you get”, se dearrolláron los mismo shaders tanto en UE3 como en Softimage, de esta forma veían el resultado final directamente en el editor de modelado.


Autodesk Softimage 2013 SP1 (64-bit) Project: AXL Scene: AXL_MODEL140620_4gamer_normal_a
MTMR_Toonshader
4Gamer.net
http://www.4gamer.net/

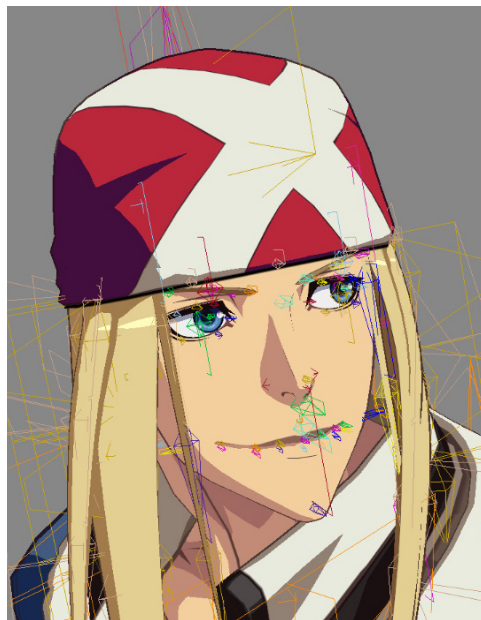

Ship_PL - Unreal Editor for GuiltyGearXrd (32-bit, DX9)
Unreal Material Editor: CHR_Materials.Materials.M_Character
Content Browser
half lambert
Specular
HalfVector
RED Material Link
Parameter Name
SssMap
Group
Texture
Texture2D'AXL_MAT_0101.Texture.AXL_Sss'
4Gamer.net
http://www.4gamer.net/

- Se estableció un budget para la escena de unos 800.000 polígonos, 500.000 para los fondos y 250.000 para los personajes. Actualmente los personajes rondan los 40.000 polígonos, sobre todo por la cantidad de assets pueden aparecer dinámicamente en pantalla.



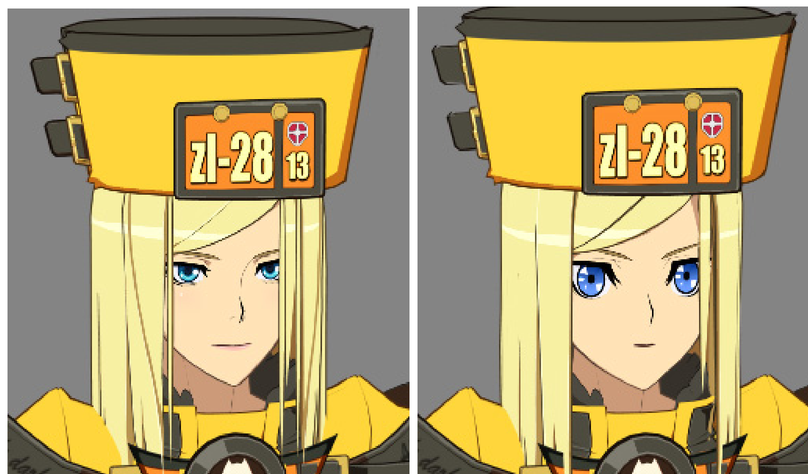
- Sol, que es uno de los personajes tiene 460 huesos, otros personajes alcanzan los 600, y de estos unos 170 se utilizan para cambiar expresiones faciales y que se vean más como en 2D. El resto cabe pensar que es para cada pieza de la ropa, cada cinturón etc.. tenga animaciones, es decir, no utilizan físicas de ropa para animar la ropa sino que también son animaciones con huesos, esto es lo que entiendo dificulta mucho que a un personaje se le puedan cambiar partes del vestuario más allá de los colores de la ropa, un personaje con un traje diferente es directamente otro personaje, con otro esqueleto y otras animaciones.





- Hay cabezas de personajes separadas para algunas escenas que son en plano corto o plano muy cercano. Esto podríamos pensar que se trata de LoD, level of detail, pero no es eso exactamente, no tienen por qué contener menos o más polígonos, es solo para asegurarse de que la expresión facial del personaje se sigue pudiendo apreciar con claridad independientemente de la distancia, por ejemplo, ojos más grandes, contornos más pronunciados y cosas así.

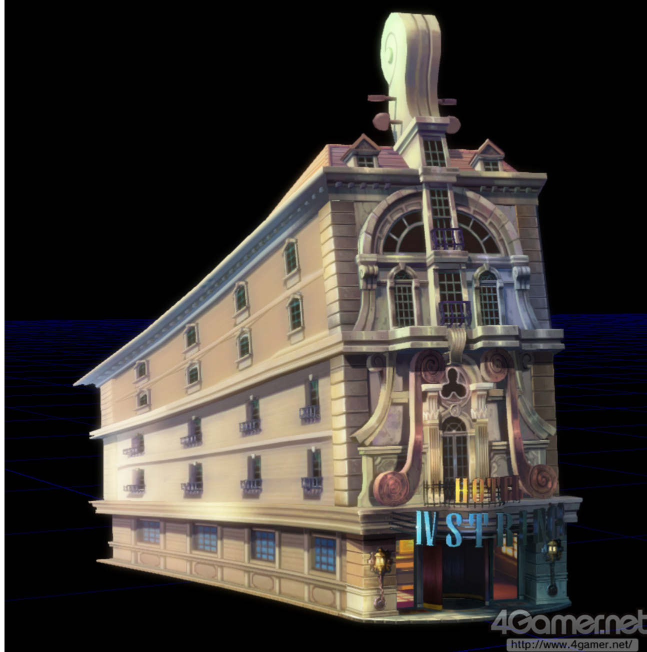


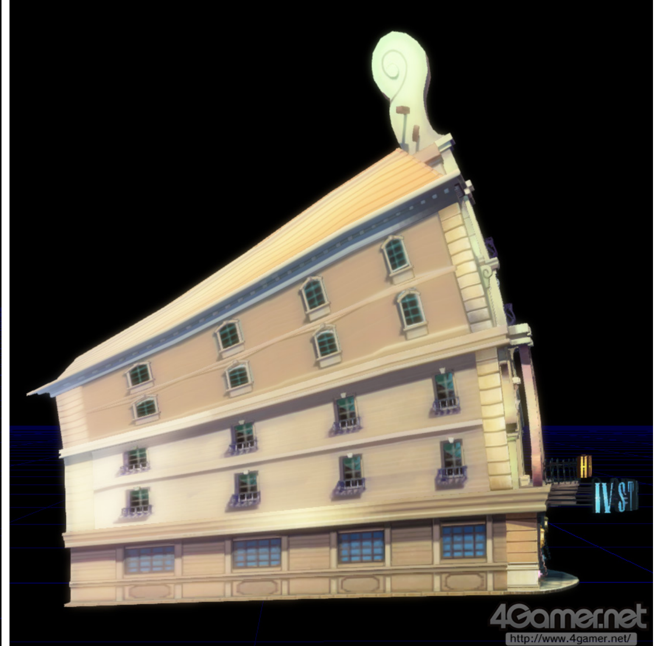


- A parte de la cabeza hay otras partes del cuerpo que también están separadas del personaje principal y que se muestran cuando el personaje realiza alguna técnica que hace que modifique parte de su modelo. Por ejemplo hay un personaje que tiene el cabello largo y hace ataques con el pelo, pues en lugar de ser animaciones estos movimientos son directamente otro modelo 3D que se pinta encima.
- En ocasiones, han requerido también fakear la perspectiva, en algunos momentos o ataques, y cuando el FOV, field of view, de la cámara no ha sido suficiente, lo que han hecho ha sido escalar los huesos en los 3 ejes. Esto por ejemplo no está permitido en UE3 por defecto, pero destacan la ventaja de haber tenido acceso al código fuente para poder implementarlo. Gracias a esta característica han realizado montones de animaciones propias de una serie de dibujos animados, como escalar las manos o los brazos a la hora de pegar un puñetazo para exagerar el golpe.
- Un punto curioso me pareció el tema partículas, en este juego hay un efecto al final del combate en el que la cámara da una vuelta de 360 grado alrededor de los personajes, esto obliga a que efectos como las partículas de humo, o nubes de humo, deben poder ser representadas correctamente en 3D y no ser un simple billboard. Tras probar con varias técnicas, al final optaron por hacerlo por fuerza bruta, cada frame de animación de las nubes de humo es una malla propia! Aunque aparentemente parece costoso para un efecto de partículas pero la verdad es que permite al artista elegir exactamente como se verá el humo. Esta técnica también la utilizan con un personaje llamado Zato que se transforma en una especie de sombra, pues la transición que se hace con un humo oscuro también la realizan con un modelo aparte.

Composición del escenario

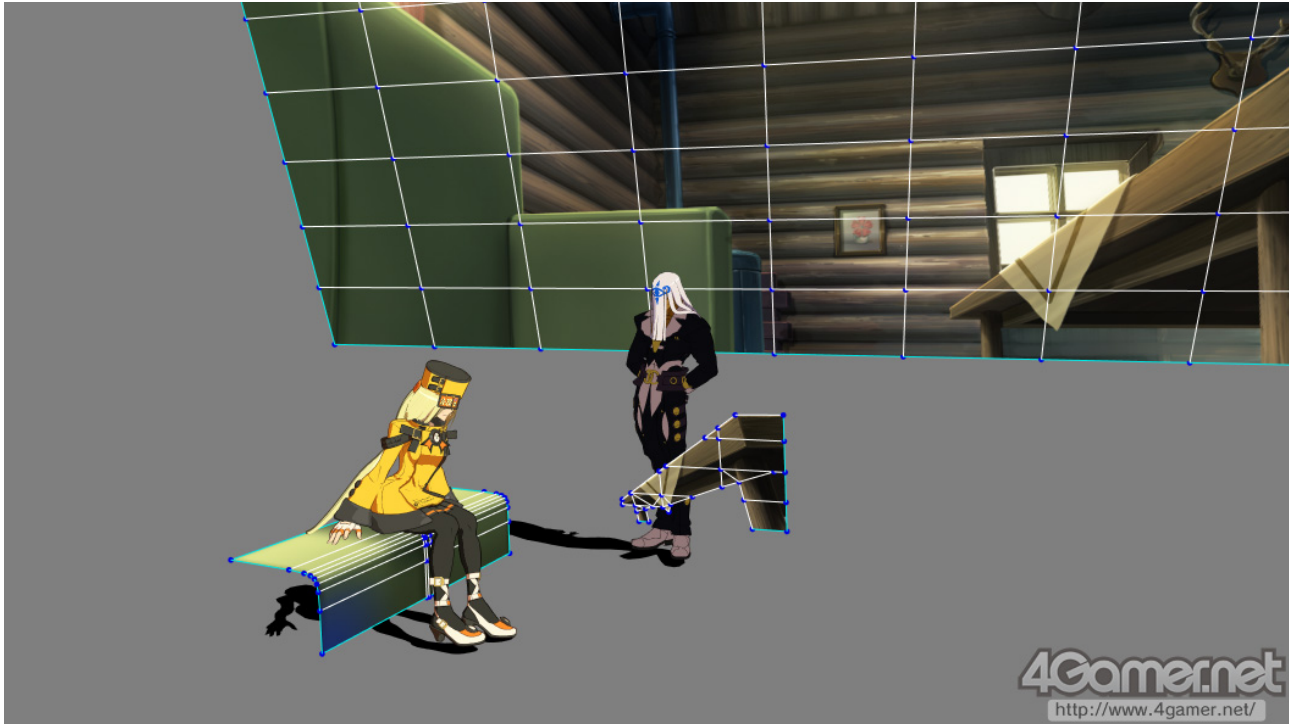
- En juegos de lucha 2D lo más habitual para el escenario es poner varias capas a diferentes profundidades para crear un efecto parallax cuando te mueves, moviendo los objetos lejanos más despacio que los objetos cercanos. Este efecto en 3D viene dado de forma natural gracias a la perspectiva, pero en Guilty Gear, como no, está masivamente falseado.
- Los modelos del escenario que parecen lejanos pueden estar más cerca de lo que parecen pero han sido modelados directamente para causar ese efecto, esto lo podemos ver en uno de los edificios que aparentemente debería ser un cubo alargado, pero en realidad la parte trasera es mucho más baja que la frontal, exagerando el efecto de la perspectiva.





- En algunas escenas cinemáticas puede ser todavía más exagerado y todo el escenario puede estar representado por una sola imagen, y los personajes pintados encima de esta, como si de un teatro de marionetas se tratara.





- Los personajes de fondo que forma parte del escenario más cercanos, son modelos 3D igual que los luchadores, pero para los personajes más lejanos se utilizan billboards, es decir, esos si que serian sprites clásicos pintados con un quad.

Animación

- Hemos hablado sobre los modelos 3D y renderizado, pero algo que traiciona bastante la percepción de si lo que vemos es 2D o 3D es también la animación de los personajes.
- Aunque el juego esté renderizado con toon shading, sigue ejecutándose a 60 fps, pero la animación que vemos en las series de TV es de unos 12 u 8 fps, de hecho, si en alguna serie de TV de animación 2D, vemos algo animado en 3D suele ser bastante cantoso, lo percibimos perfectamente.
- Para prevenir este efecto no deseado el juego está animado como si fuera a 15fps, aunque algo más complejo que eso. En TV se considera que una animación completa va a 24 fps por segundo, y una limitada como es habitual en series a 12 u 8 como ya habíamos mencionado.
- En Guilty Gear al ser un videojuego, los storyboards se basan en 60 fps para la numeración de los frames de animación pero por cada frame se especifica cuantos frames del juego han de transcurrir, creando esa media de 15 fps.
- De esta forma no estás simplemente bajando el framerate general, si no que además estás eliminando la interpolación que habria entre esos frames. Básicamente la animación se hace de una pose a otra sin interpolar, pero dejando que transcurra el tiempo adecuado para cada frame en lugar de un tiempo constante.
- **Ejemplo: https://www.youtube.com/watch?v=xZ3npkMNj0s**

Renderizado

- En PS3 el juego se ejecuta a 60fps en 1280x720 y en PS4 a 60fps 1080p
- Se utiliza Forward Rendering en lugar del Deferred.
- Cada personaje se renderiza 4 veces, una para la Z pre pass y el color, otra para el outline y otra para la mesh final.
- En cada frame, se utilizan unos 160mb de información de textura y entre 70-80 shaders, todo creados en UE.
- La sombra de los fondos no es real, todas las sombras han sido pintadas y solo se utiliza una luz en toda la escena utiliza para castear la sombra de los personajes en el suelo.
- Los personajes ni siquiera reciben esta luz, utilizan un conjunto de luces dinámicas locales que solo les afecta a ellos y que animan cada frame de la animación para que el personaje se vea lo mejor posible.



- La base del cel shading que usan es la clásica, se establece un margen de cantidad de luz, si se sobrepasa está iluminado, si no, oscurecido, pero obviamente hay una serie de trucos para que no sean los polígonos los que mandan sobre este sombreado, sobre todo para evitar pequeñas sombras una zona iluminada o pequeños puntos luminosos a lo largo de una zona sombreada.
- Para conseguir utilizan información guardada tanto en texturas como en los atributos del vértice de la maya.
- En el canal rojo del atributo color del vertice guardan un shadow bias, en este caso viene a ser como un ambient oclusion pero en lugar de generado, está indicado por el artista, cuanto más bajo sea este valor en este vértice, más propenso será a sombrearse, cuanto más alto, más propenso a iluminarse.







- Las normales de los polígonos también están tuneadas con este propósito, por supuesto, aquí tuvieron un problema y es que al exportar de softimage a ue3 esta información se perdía y al final recurrieron a exportar otra maya cuyas normales por defecto ya fueran las que necesitaban y reemplazar el atributo de la normal del vértice de la malla original en el propio engine.
- Otro elemento para controlar la iluminación es una textura a la que han llamado ILM Texture.
- En el canal verde de esta textura, hay otro shadow bias, en este caso para fakear áreas que están siempre iluminadas o siempre sombreadas, essencialmente para falsear las self shadows, las sombras que un personaje deberia castear sobre si mismo, no las hace el motor si no que se basan en esta información, valor indica la intensidad.
- En el canal Rojo de la ILM texture se guarda la intensidad de la luz especular y en el canal Azul el specular size.
- El color de la sombra está controlado por otra textura, llamada SSS texture, básicamente en una zona de luz el color será, (color de la luz + color ambiente) multiplicados por el color de la textura del personaje, en una zona sombreada es color ambiente por color del personaje por SSS color, de esta forma tintan la sombra según la zona del personaje. A mi esto del SSS me llevó a pensar una cosa que comentan y es que SSS se puede interpretar como sub-surface scattering, aplicable en superficies transparentes pero con opacidad debajo como un bloque de hielo o de cera pero que por lo que dice, simplemente escogieron ese nombre y que no tiene nada que ver.





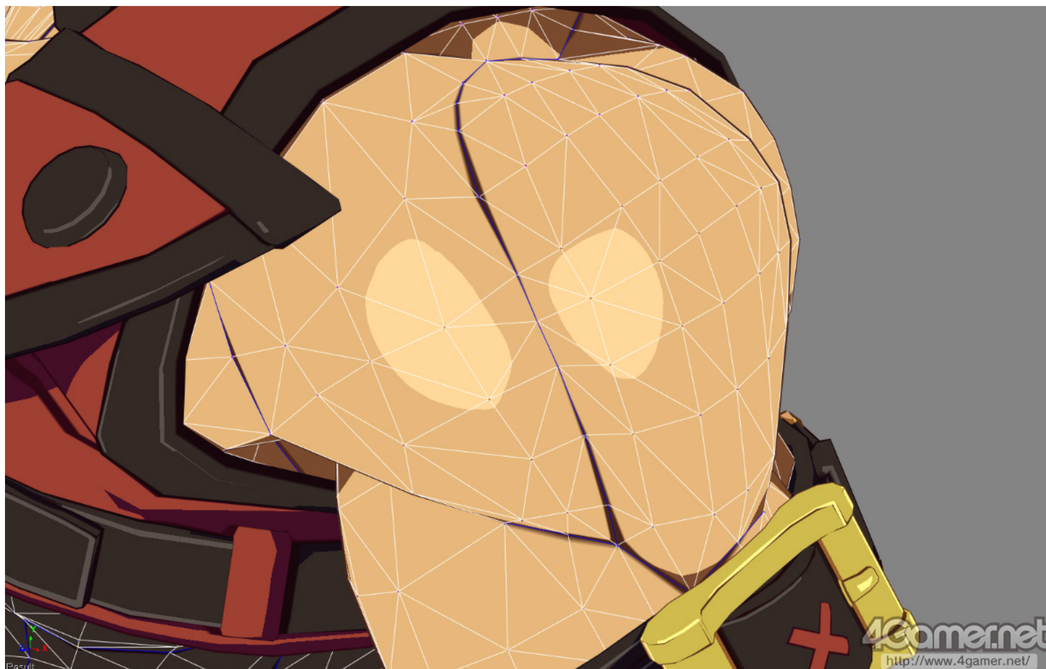
- Los bordes de los personajes, como ya hemos dicho están hechos renderizando el personaje como una silueta oscura pero con una escala ligeramente mayor que el original y luego renderizando el personaje encima,
- Aunque esto también está tuneado, en el canal alpha del atributo de color del vértice, se guarda el grosor o densidad el borde en ese vértice, en el verde se guarda cuánto escala según la distancia de la cámara, y en el azul un offset en Z utilizado para llevar el vertice más atrás en el Z buffer, haciéndolo invisible cuando está en frente de un objeto cercano, esto es muy útil por ejemplo en el pelo, para que no haya muchos bordes intercalads por cada mechón de pelo, si no que solo los mechones que más sobresalgan tengan borde, piensa en el pelo de son goku cuando lo dibujan con poco detalle en el anime y ponen bordes en cada mechón, pues eso.



- Eso sirve para contornos externos, pero para lo que son bordes internos de músculos, rasgos faciales, arrugas en la ropa, en lugar de pintar las líneas directamente en la textura, para evitar bordes aserrados debido a la limitación de tamaño en la textura, lo que hacen es, para saber donde hay un borde utilizan el canal Alpha de la ILM texture, aquí se pintan todos los bordes pero están estrictamente alineados con el eje x o y, osea no hay ninguna línea en diagonal, con lo cual no pueden aparecer con borde aserrado. Luego utilizan una técnica propia a la que

han llamado “Motomura lines”, líneas de Motomura, que es el lead modeler y tech artist, y consiste básicamente en utilizar la topografía de las coordenadas de textura para decidir la dirección y grosor de estas líneas, incluyendo el inicio y el final de estas, de esta forma estos bordes internos nunca aparecen aserrados o pixelados independientemente de la distancia.





Autodesk Softimage 2013 SP1 (64-bit) Project: SOL Scene: SOL_MODEL_140325_cgworld_line_c

Texture Editor

Camera

Result

Select Components

Autodesk Softimage 2013 SP1 (64-bit) Project: SOL Scene: SOL_MODEL_140325_cgworld_line_c

Texture Editor

Camera

Result

Select Objects

Perspectiva y Hit Boxes

- Al principio, cuando hemos hablado del desarrollo habíamos comentado que habían portado herramientas como la que les permite animar las hit boxes, pero hay un problema con las hit boxes en un juego 3D.
- Pese a que el sistema sigue funcionando muy bien ya que la lucha transcurre en un plano 2D, la perspectiva en 3D crea una ligera distorsión en el modelo, técnicamente llamada shear, que hace que cuanto más alejado está el modelo 3D del medio de la pantalla, menos encaje con sus hit box que se encuentran en un plano 2D no afectado por la perspectiva. Concretamente el modelo del personaje es más ancho que su hit box.
- Para que los personajes estén correctamente representados por las hit boxes a la hora de calcular los golpes, lo que se hace es fakear la perspectiva, en lugar de utilizar parallel projection, lo que solemos llamar renderizado Orthonormal o Anamórfico, hacen un híbrido 30% perspective y 70% parallel, está técnica ya había sido utilizada en Street Fighter 4 de hecho. Aquí la han utilizado únicamente en el eje horizontal, el eje vertical es 100% perspective ya al ser la pantalla más ancha que alta, los personajes en vertical no se distorsiona demasiado.

Otros trucos de imitación de 2D

- Debido a que los modelos son 3D es habitual que al estar juntos se solapen entre ellos y se crucen ya que la prioridad que pixel va primero se basa en la información del Z buffer, en lugar de en qué orden fueron enviados a pintar como suele hacerse en un juego 2D donde se omite la Z.
- Para solucionar esto lamentablemente no se puede simplemente poner un personaje delante del otro porque la perspectiva haría que el más cercano se viera mas grande que el de atrás, en lugar de eso lo que hacen es aplicar en uno de los dos personajes (desconozco el criterio para elegir cual) un offset en el valor de profundidad de los pixeles después de aplicar la perspectiva, afectando este valor únicamente al z-buffer y no a la rasterización, con lo que su tamaño quedará intacto pero la parte su cuerpo más cercana al otro personaje aparecerá detrás.
- Otro truco interesante, el tema de girar el personaje, en un juego 2D normalmente lo que hacemos es hacer flip del sprite, invertir la escala en el eje x, y curiosamente en Guilty Gear han hecho lo mismo con los modelos 3D, cambiando el culling order para indicar que lado del triángulo es el correcto.
- Pero al hacer esto nos encontramos otro problema brillantemente solucionado, algunos personajes en al ropa tienen símbolos y texto que no pueden ser girados haciendo mirror ya que se leerían al revés, para solucionar esto todos estos textos y símbolos no están en la textura del personaje sin que son decals, osea, una malla con la textura que contiene el texto o símbolos proyectada sobre el modelo, así al invertir el eje X, simplemente se invierten las UVs de la decal, así el texto no aparece invertido.

Efectos de post-procesado

- El antialiasing es FXAA en lugar del MSAA.
- Se aplica bloom a todo lo que tenga un valor de luminancia mayor de 1.0
- Y un diffusion filter, que suaviza un poco más la escena, y da un efecto similar al de los dibujos que vemos por la tele en el que las zonas más claras irradian los pixeles de zonas más oscuras. Este efecto también se puede ver en el juego de Skull Girls.



